Программирование

Python

Модули python

Кафедра ИВТ и ПМ

2018

План

Прошлые темы

Модули

- Подключение модулей
- Создание модулей
- Отношения между модулями

Model-View-Controller

Outline

Прошлые темы

- Что такое глобальные переменные?
- Что такое локальные переменные?
- Чем отличаются формальные параметры от фактических?
- Что такое чистая функция?
- Почему не рекомендуется использовать глобальные переменные внутри функций?

Прошлые темы

- Что такое парадигма программирования?
- Что такое структурное программирование?
- Что такое процедурное программирование?
- В чём смысл принципа "не повторяй себя"(dont repeat yourself, DRY)?

Проблема?

Программа 1

```
L = [1.0823, 2.2221, 3.872]

for e in L:
    print("{:.2}".format(e), end="")

for i in range( len(L)):
    L[i] = L[i]**2

for e in L:
    print("{:.2}".format(e), end="")
```

Проблема?

Программа 1

```
L = [1.0823, 2.2221, 3.872]

for e in L:
    print("{:.2}".format(e), end="")

for i in range( len(L)):
    L[i] = L[i]**2

for e in L:
    print("{:.2}".format(e), end="")
```

Код для вывода списка вещественных чисел на экран повторяется.

Решение

Программа 1

```
def print_list(L):
    for e in L:
        print("{:.2}".format(e), end="")

L = [1.0823, 2.2221, 3.872]
print_list(L)
for i in range( len(L)):
    L[i] = L[i]**2
print_list(L)
```

Проблема?

Программа 2

```
def print_list(L):
    for e in L:
        print("{:.2}".format(e), end="")

L = [1.0823, 2.2221, 3.872]
L2 = []
print_list(L)
for x in L:
    L2 += sin( e )
print_list(L)
```

Проблема?

Проблема?

Программа 2

```
def print_list(L):
    for e in L:
        print("{:.2}".format(e), end="")

L = [1.0823, 2.2221, 3.872]
L2 = []
print_list(L)
for x in L:
    L2 += sin( e )
print_list(L)
```

Проблема?

Для использования функции в другой программе она была скопирована. Две программы содержат одинаковый код.

Outline

Модуль

Модуль — функционально законченный фрагмент программы. Во многих языках оформляется в виде отдельного файла с исходным кодом или поименованной непрерывной её части.

Модули могут объединятся в **пакеты**.

Модульное программирование

Модульное программирование — это парадигма программирования, согласно которой программа строится из небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам.

Преимущества модулей

- Структурирование кода.
- Повторное использование кода.
- Сокрытие сложности.
- Использование модулей написанных на другом языке программирования.

Outline

Подключение модулей

- `import <имя_модуля>`

```
# пример
import math
math.sin( math.pi )
sin(0)    # name 'sin' is not defined
```

- `import <имя_модуля> as <новое_имя>`

```
# пример
import math as m
m.sin( m.pi )
math.sin(0)  # name 'math' is not defined
```

Подключение модулей

- `from <имя_модуля> import имя1, имя2`

```
# пример
from math import pi, sin
sin( math )
cos( 0 )  # name 'cos' is not defined
```

- `from <имя_модуля> import *`

```
from math import *
sin( pi ) + cos( e )
math.sin(0)  # name 'math' is not defined
```

Подключение модулей

Операция подключения модуля выполняется только один раз. Даже если вызывать её повторно.

```
import math

print(math.pi)  # 3.141592653589793
math.pi=4

import math
print(math.pi) # 4
```

Изменять переменные модулей не рекомендуется

Подключение модулей

Если подключаемый модуль недоступен то программа завершается с ошибкой **ModuleNotFoundError**

```
import godmodule
```

```
ModuleNotFoundError: No module named 'godmodule'
```

Подключение модулей

Чтобы проверить[1] доступность модуля можно использовать функцию find_spec из модуля importlib.util

Эта функция по заданному имени модуля возвращает информацию о модуле (значение типа ModuleSpec).
Если информация не найдена, то функция возвращает None.
Это означает что модуль недоступен.

```
import importlib
spec = importlib.util.find_spec("module_name")
if spec is None:
    print("Модуль не существует")
```

[1] см. StackOverflow

Подключений модулей

Алгоритм действия транслятора при подключении модуля

1. Поиск файла модуля
   имя_файла = имя_модуля + .py
   имя_файла = имя_модуля + .pyc
   1.1 поиск в текущем каталоге
   1.2 поиск в местах перечисленных в переменной PYTHONPATH
   1.3 поиск в каталогах стандартной библиотеке
   1.4 поиск в местах приведённых в .pth файлах[2]
2. Компиляция модуля в байт-код[3] (py -> pyc) при необходимости
3. Подключение (запуск) модуля
   Интерпретатор выполняет все инструкции в модуле, например оператор определения функции def

[2].pth файлы находятся в каталоге Python

[3]промежуточное представление программы между исходным кодом и машинным; байт-код выполняется виртуальной машиной, а не процессором. Виртуальная машина для выполнения кода python входит в состав python

Пути поиска

Пути поиска файлов записаны в переменной **path** модуля sys

```
import sys

print(sys.path)

# вывод для ОС Debian
['',
'/usr/lib/python36.zip',
'/usr/lib/python3.6',
'/usr/lib/python3.6/lib-dynload',
'/usr/local/lib/python3.6/dist-packages',
'/usr/lib/python3/dist-packages',
'/usr/lib/python3/dist-packages/IPython/extensions']
```

Пути поиска

При необходимости при запуске программы можно добавить дополнительный путь поиска модулей

```
import sys
# добавление пути поиска в самое начало списка
sys.path = ['/home/user/my_modules'] + sys.path

# теперь можно подключить модуль
# с расположением /home/user/my_modules/test_m.py
import test_m
test_m.foo()
```

Outline

Создание модулей

Модуль в Pyhton - это отдельный файл с расширением py.

Отличия модуля от программы - в способе их использования. Программа предназначена для того, чтобы её запускали, а модуль - чтобы его использовали в программе.

Синтаксической разницы между программой на языке Python и модулем нет.

Поэтому можно подключить любую программу как модуль.

Создание модулей

Модули в отличии от программ часто не содержат исполняемых при запуске (подключении) операторов.

Модуль как правило состоит из набора подпрограмм, классов и переменных.

Создание модулей

Основная программа

```
from my_module import *

L = [1.0823, 2.2221, 3.872]
L2 = []
print_list(L)
for x in L:
    L2 += sin( e )
print_list(L2)
```

Модуль *my_module.py*

```
def print_list(L):
  """ Выводит на экран список
  вещественных чисел
  """
  for e in L:
    print("{:.2}".format(e), end="")
```

Имена модулей

Имя модуля совпадает с именем файла, не считая расширения.

Стоит внимательно выбирать собственных имена модулей, так чтобы они не перекрывали имена установленных в python модулей.

Области видимости

Каждый модуль имеет свою область видимости.

Если идентификатор в главной программе и в модуле совпадают, то будет использован идентификатор из главной программы.

Идентификаторы в двух модулях могут совпадать. Поэтому не следует подключать всё содержимое больших модулей если есть вероятность конфликта имён.

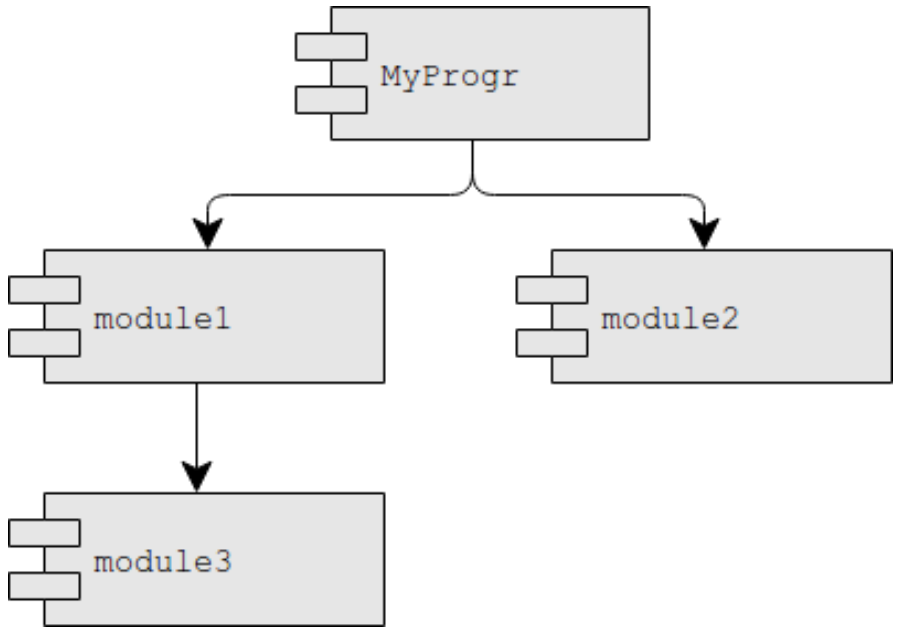
Outline

UML

UML (Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

Диаграмма модулей является частным случаем диаграммы компонентов в UML.

Диаграммы модулей



На диаграмме модулей стрелочки направляются от основного модуля, к вспомогательным.

Связность

Связность, или прочность (cohesion, module strength), — мера силы взаимосвязанности элементов внутри модуля.

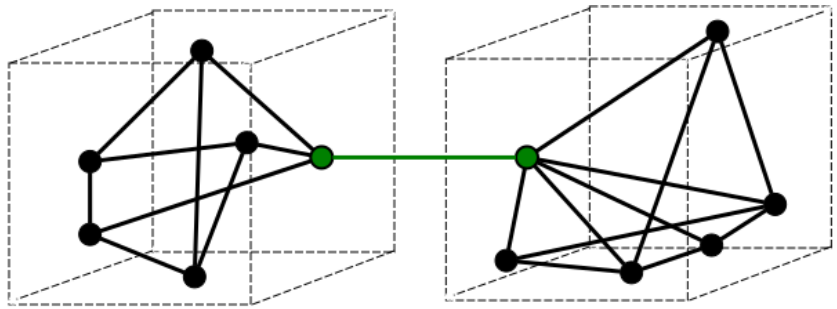
связность характеризует то, насколько хорошо все части модуля соответствуют главной цели (назначению) этого модуля.

Зацепление

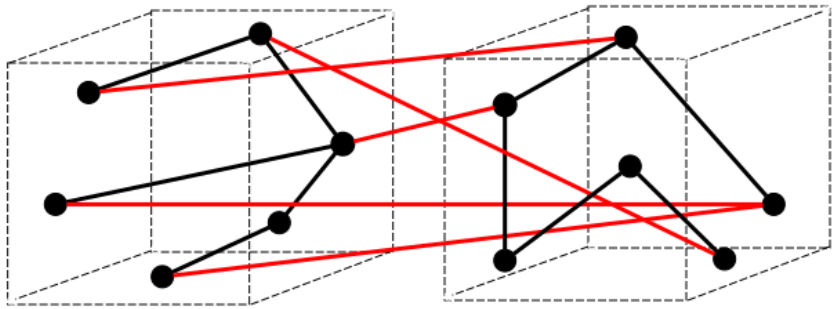
Зацепление, сцепление, сопряжение (coupling) — способ и степень взаимозависимости между модулями.

Сильное зацепление рассматривается как серьёзный недостаток, поскольку затрудняет понимание логики модулей, их модификацию, автономное тестирование, а также переиспользование по отдельности.

Связность и зацепление



a) Слабое зацепление, сильная связность



b) Сильное зацепление, слабая связность

Слабое зацепление является признаком хорошо структурированной и хорошо спроектированной системы, и, когда она комбинируется с **сильной связностью**, соответствует общим показателям хорошей читаемости и сопровождаемости.

Закон Деметры

Закон Деметры[4] (Law of Demeter, LoD) — набор правил проектирования при разработке программного обеспечения.

Каждый программный модуль:

- должен обладать ограниченным знанием о других модулях: знать о модулях, которые имеют «непосредственное» отношение к этому модулю.
- должен взаимодействовать только с известными ему модулями «друзьями», не взаимодействовать с незнакомцами.
- обращаться только к непосредственным «друзьям».

[4] назван в честь проекта Деметра, где широко использовались основные идеи закона

Outline

Model-View-Controller

Model-View-Controller (MVC, Модель-Представление-Контроллер, Модель-Вид-Контроллер) — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо

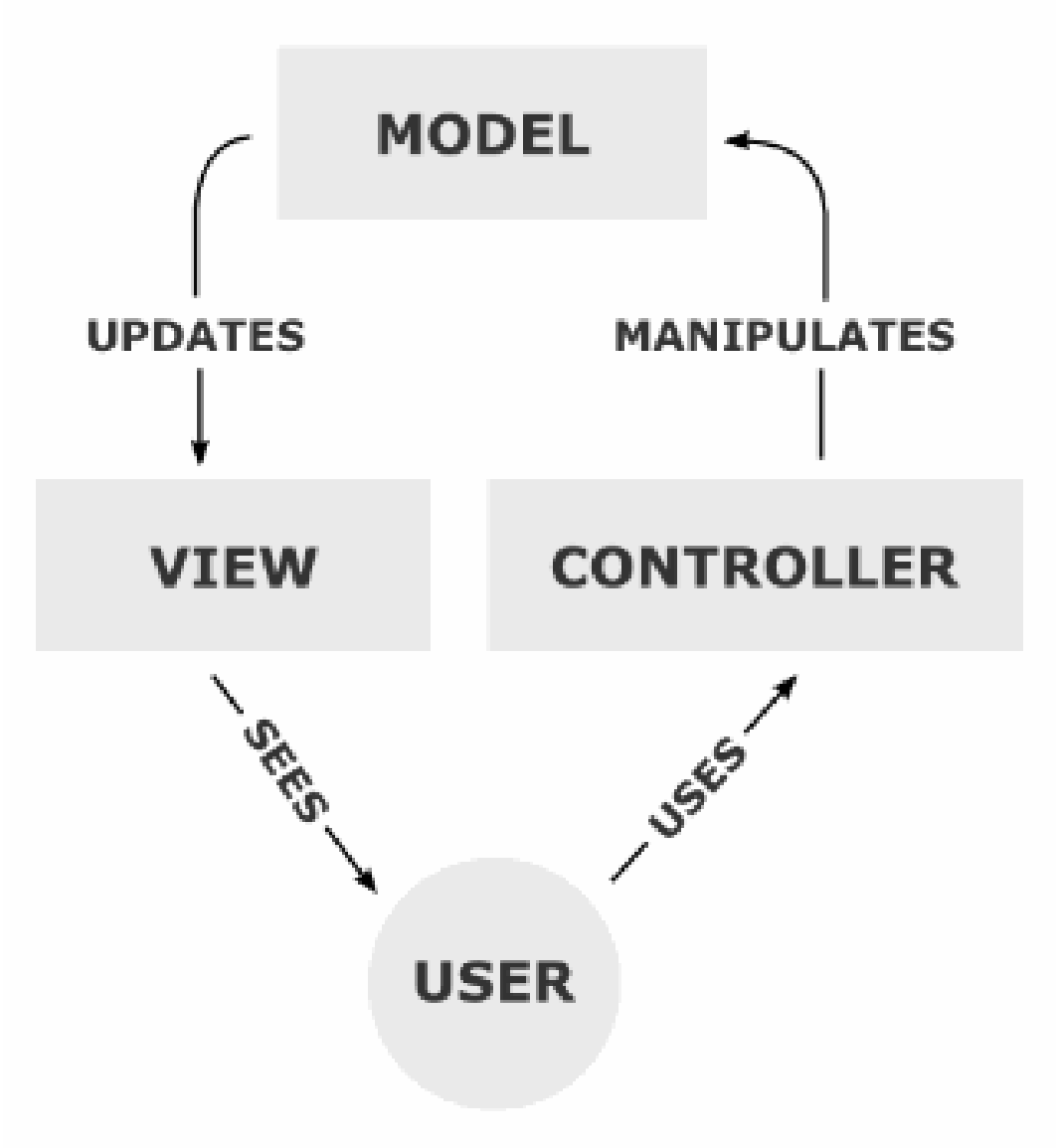
Модель

Модель - представление некоторого реального процесса, устройства или концепции.

Model-View-Controller

- **Модель** (Model) предоставляет данные и реагирует на команды контроллера, изменяя свое состояние.
- **Представление** (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
- **Контроллер** (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений

Model-View-Controller



Model-View-Controller

Модель представляет собой **бизнес-логику**.

Бизнес-логика (логика предметной области) — совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает).
Примером логики предметной области могут быть правила игры в шахматы.

Правила не меняются в зависимости от того как представлены доска и фигуры и как реализован способ их передвижения. Фигуры могут быть трёхмерными моделями, а их передвижение реализовано с помощью мыши. Если представить игру двумерной и управлять передвижением фигур указывая старые и новые координаты на доске, сами правила игры в шахматы не поменяются.

Ссылки и литература

- draw.io - сервис для создания диаграмм

Ссылки и литература

Ссылка на слайды

github.com/VetrovSV/Programming